Negociation/argumentation techniques among agents complying to different ontologies

Jérôme Euzenat, Loredana Laera, Valentina Tamma, Alexandre Viollet

D2.3.7: Negotiation/argumentation techniques among agents complying to different ontologies

Coordinator:
Jérôme Euzenat (INRIA Rhône-Alpes),
Loredana Laera (University of Liverpool),
Valentina Tamma (University of Liverpool),
Alexandre Viollet (INRIA Rhône-Alpes)

Abstract.
This document presents solutions for agents using different ontologies, to negotiate the meaning of terms used. The described solutions are based on standard agent technologies as well as alignment techniques developed within Knowledge web. They can be applied for other interacting entities such as semantic web services.
Keyword list: ontology matching, ontology alignment, agents, negotiation, meaning, ontology alignment, protocol, dialog, argumentation.

Document Identifier	KWEB/2004/D2.3.7/v1.0
Project	KWEB EU-IST-2004-507482
Version	v1.0
Date	February 6, 2006
State	final
Distribution	knowledge web

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale
Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer
Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

Centre for Research and Technology Hellas
Free University of Bozen-Bolzano
Institut National de Recherche en Informatique et en Automatique
National University of Ireland Galway
University of Innsbruck
University of Karlsruhe
University of Liverpool
University of Manchester
University of Sheffield
Vrije Universiteit Amsterdam
Vrije Universiteit Brussel

Changes

Version	Date	Author	Changes
0.1	31.01.2005	Jérôme Euzenat	creation
0.2	22.04.2005	Jérôme Euzenat	added the protocol rules
0.3	28.04.2005	Jérôme Euzenat	added introduction
0.4	28.06.2005	Jérôme Euzenat	translated/put in 2.3.7/expanded
0.5a	26.07.2005	Jérôme Euzenat	commented 3.5/improved 5
0.5b	03.09.2005	Loredana Laera & Valentina Tamma	added framework
0.6	19.09.2005	Jérôme Euzenat	merged version 0.5
0.61	24.09.2005	Jérôme Euzenat	slightly improved 5 (and 3.4.3)
0.7	11.10.2005	Loredana Laera	improved 5 and added 5.5.1
0.8	22.11.2005	Jérôme Euzenat	overall review, achieved section 3, reorganized section 5, added executive summary
0.8a	25.11.2005	Jérôme Euzenat	suppressed chap 4, completed 2.6, completed conclusion
0.9	14.12.2005	Valentina Tamma	revised state of the art adding the difference between bargaining and argumentation
1.0	03.02.2006	Valentina Tamma, Loredana Laera & Jérôme Euzenat	addressed reviewers comments

Executive Summary

The goal of this deliverable is to investigate the means that independently designed agents, using different ontologies, have to interoperate.

The first easy answer is to standardize the ontologies so that they use the same ones. This answer is suitable to very constrained environments (for instance when a customer can impose standards to providers). However, the provider that has several customers will have to solve the interoperability problem anyway.

The second solution consists of considering that there exist libraries of alignments or trustable alignment services that can be invoked in order to obtain an alignment between two ontologies and use this alignment for translating messages. Therefore, in this deliverable, we investigate, given two ontologies used by two agents, how to find an alignment between these ontologies that can be used for exchanging messages, for instances. We draw on the work done in Knowledge Web work package 2.2 in order to deal with the following issues:

– how agents can take advantage of alignment services for obtaining and using alignments;
– how agents can negotiate the content of an alignment.

First, we present a protocol allowing agents to obtain such an alignment that suits their needs (Chapter 3). This protocol allows agents to find similar ontologies, to ask for alignment between ontologies using various methods, to ask for the translation of messages according to existing alignments and to generate translation programs or mediators that can be used by the agents themselves.

But what if it is not possible to obtain an alignment that suits both parties? It is then necessary for these parties, if they want to interact, to negotiate the meaning of terms, or, more modestly, to negotiate the correspondences in alignments. For that purpose, we introduce a novel argumentation framework for arguing for and against correspondences found in alignments (Chapter 4). This framework is based on previous work on argumentation in multi-agent systems but adapts it to the specific case of arguing about alignments and correspondences. It provides a first typology of arguments (local, global and preferences) that can be applied to correspondences between ontology entities. A number of concrete arguments are provided. They are then used in an argumentation protocol, whose design required a modification of classical argumentation logic so that it can accomodate alignments. We provide strategies for evaluating arguments during the unfolding of the negotiation dialogue.

The work presented in this deliverable is especially designed to be used by agents of any type, primarily software agents, but also any kind of clients (even human users) and web services (where it helps creating mediators for semantic web services).

Contents

Chapter 1

Introduction

The goal of this deliverable is to investigate how independently developed agents, committing to different ontologies, and situated in a dynamic environment can achieve agreement on the type of vocabulary to use in order to interoperate.

The first easy answer is to standardize the ontologies so that they use the same ones. This answer can hold in very constrained environments (for instance when a customer can impose standards to providers). However, the provider which has several customers will have to solve the interoperability problem anyway.

The second solution consists of considering that there exist libraries of alignments or trustable alignment services that can be invoked in order to obtain an alignment between two ontologies and use this alignment for translating messages for instance. We draw on the work done in Knowledge Web work package 2.2 in order to present a protocol allowing agents to obtain such an alignment that suits their needs (Chapter 3). The alignment service is designed along the lines of Deliverable 2.2.6.

But what if it is not possible to obtain an alignment that suits both parties? It is then necessary for these parties, if they want to interact, to negotiate the meaning of terms, or, more modestly, to negotiate the correspondances in alignments. For that purpose, we introduce a novel argumentation framework for arguing for and against correspondences found in alignments (Chapter 4). This framework is based on research efforts in agent negotiation based on argumentation, that were reviewed in Deliverable 2.3.2.

Hence, the problem that we consider is, given two autonomous agents committed to two ontologies O and O', how can they find an agreement on the correspondence between the vocabulary they use (this will be expressed as an ontology alignment). It is noteworthy that the process of reaching agreement should be as automatic as possible and should not require any feedback from human users.

In order to achieve this goal, we take advantage of tools developed in the area of multi-agent systems and ontology matching and we adapt them to this problem. The specific point of this deliverable with regard to the work done in work package 2.2, is that it deals with the following issues:

- how agents can take advantage of alignment services for obtaining and using alignments (for that purpose, we define a communication protocol based on agent standards);
- how agents can negotiate the content of an alignment (for that purpose we define an argumentation framework adapted to correspondence negotiation).

The work presented in this deliverable is especially designed to be used by agents of any type, primarily software agents, but also any kind of clients (even human users) and web services (where it helps creating mediators for semantic web services).

The deliverable is organised along the following lines: first we consider previous attempts to bring meaning negotiation (or ontology alignment) to agents (Chapter 2); then we introduce our alignment communication protocol (Chapter 3); and finally we present a new argumentation framework for correspondences (Chapter 4).

Chapter 2

State of the art

There are relatively few works on the topic of autonomous entities negotiating about meaning in a dynamic way. We present below some of these efforts. However, beforehand, we introduce some of the techniques that will be used later on in this deliverable. We thus start with agent infrastructure (Section 3.1) and ontology alignment (Section 2.2) before presenting other work addressing the same problem.

2.1 Agent infrastructure

Research on agents and multi-agent systems has already developed many tools and techniques for modelling agents operating in isolation and within a system. Here we concentrate on agents situated in a system, that need to display *social ability* and communicate in order to carry out some task. We therefore present below some notions that will facilitate putting the remainder of this document in context.

Each agent has a name, a role and some internal store (also called a knowledge base). In some agent models, the *basic knowledge base* of an agent a may be consist of a set of beliefs, a set of desires and a set of intentions (hence the acronym BDI). However, for the purpose of this deliverable we do not need to distinguish between beliefs, desire and intentions, and we will simply assume that an agent has a knowledge base where it stores facts about the domain it knows and the environment is situated in, including knowledge about other agents (which can correspond to an *ontology*). In this framework, we also do not make explicit use of the agent role.

Agents act independently of each other, but they can interact with others through exchanging explicit messages (see below) or through modifying their environment. In particular, we will later make use of a "commitment store" which records, and makes visible to others, some knowledge that the agent has publicly endorsed.

2.1.1 Agent communication languages

Agent communication languages are designed for expressing messages that agents have to exchange. The purpose of standardising such a language is to be sure that any pair of independently designed agents can understand each other.

These languages are based on speech-act theory in which the *illocutory force* of a message can be understood independently from its *propositional content*. In short, you are able to understand

that some agent is telling you to do something (illocutory force=order or demand) even if you do not understand what (propositional content).

Agent communication languages are based on this type of separation, in which messages are characterized by their performatives (expressing the illocutory value).

One notable such language was KQML (Knowledge Query and Manipulation Language) [Finin *et al.*, 1993] and its current alternative, FIPA ACL (FIPA Agent Communication Language) [FIPA0061, 2002; FIPA0037, 2002] (the two languages have similar syntax, but different semantics, as explained in deliverable D2.4.3). Here we refer only to the FIPA ACL, but similar considerations can be applied to KQML. A message in FIPA ACL can be described as:

```
(inform
    :sender Valentina
    :receiver Terry
    :content "<Glass rdf:about='#myGlass'><quantity>0</quantity></Glass>"
    :language OWL
    :ontology http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine )
```

As can be noted, the agent communication language does not commit to some particular language or ontology: the sender has to indicate the language and ontology used.

The fields defined by FIPA-ACL are given in Table 2.1

Field name	Type	Meaning
performative	see Table 2.2	Illocutory value of the message
sender	agent id	Emitter of the message
receiver	agent id	Receiver of the message
reply-to	agent id	An agent to which the answer should be sent
protocol	protocol id	Protocol under which this message is issued
conversation-id	conversation id	Conversation in which this message is involved
reply-with	message id	Identifier with which the message must be answered
in-reply-to	message id	Identifier with which the message is answered
reply-by	date	Date by which the message must have been answered
content		Propositional content of the message
language	language id	Language in which the content is expressed
encoding	encoding id	Encoding of the content
ontology	ontology id	Ontology in which the content is expressed

Table 2.1: FIPA-ACL message fields

FIPA-ACL defines 22 performatives based on four basic ones (inform, request, confirm and disconfirm). These can be organized in 5 categories (informing, querying, negotiating, acting and error reporting). They are reported in Table 2.2.

2.1.2 Agent communication protocols

As can be seen from above, messages can be part of protocols that describe more precisely what is the context of message exchange and what is their more precise meaning in that context. Such protocols specify in particular what actions may or must an agent perform (updating its beliefs,

Performative	Category
inform	informing
inform-if	informing
inform-ref	informing
confirm	informing
disconfirm	informing
query-if	querying
query-ref	querying
subscribe	querying
call-for-proposals	negotiating
propose	negotiating
accept-proposal	negotiating
reject-proposal	negotiating
agree	acting
cancel	acting
propagate	acting
proxy	acting
refuse	acting
request	acting
request-when	acting
request-whenever	acting
failure	error
not-understood	error

Table 2.2: FIPA-ACL performatives

replying, performing some physical action, etc.) when receiving a message in the context of a conversation.

The most famous agent communication protocol is the *contract net* [Smith, 1980] that describes how bidding is usually performed (a call for bids is issued, proposals are emitted in reply, and after evaluation, agents are notified of the outcome and can commit to do the work if they are the winner).

There were no particular protocol definition languages until recently. Such protocols are now often expressed in AUML [Huget *et al.*, 2003], an agent-centered extension of UML (so-called) sequence diagrams (see Figure 2.1).

2.2 Ontology matching

We briefly introduce the work made in Knowledge Web work package 2.2 on heterogeneity and more specifically on ontology matching [Bouquet *et al.*, 2004; Euzenat *et al.*, 2004].

2.2.1 Matching process

Ontology matching is the process depicted in Figure 2.2. From a pair of ontologies (o and o'), it generates a set of correspondences between these ontologies called an alignment (A'). The

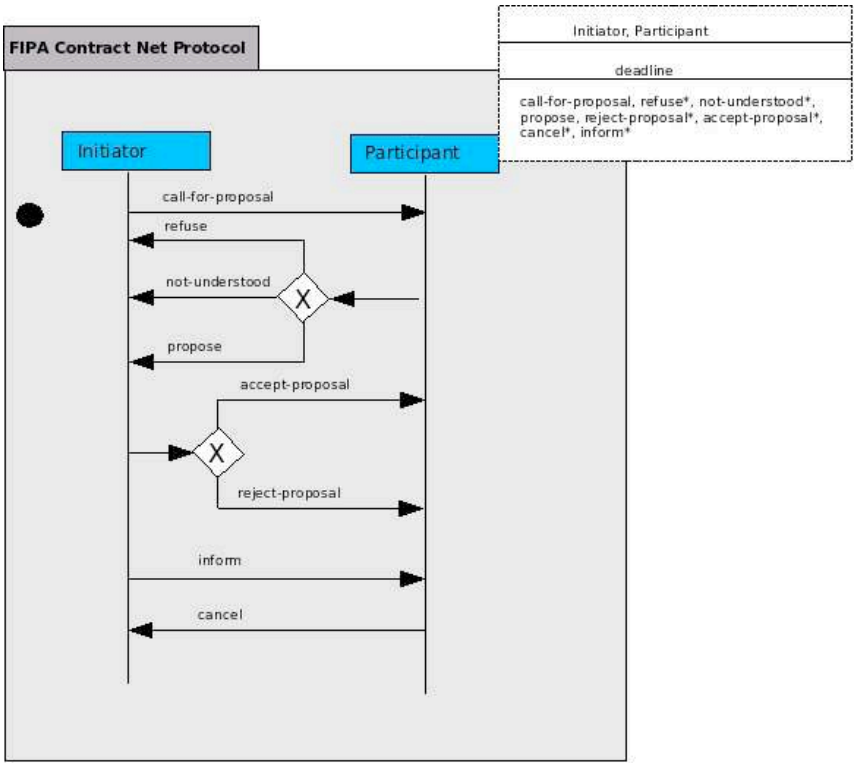

Figure 2.1: The contract-net protocol specified in AUML.

process is, of course, fully dependent on the matching method (m). However, there are various other parameters which can extend the definition of the matching process. These are namely, the use of an input alignment (A) which is to be completed by the process, the method parameters (which can be weights for instance) and some external resources used by the process (which can be general-purpose resources not made for the case under consideration, e.g., lexicons, databases).

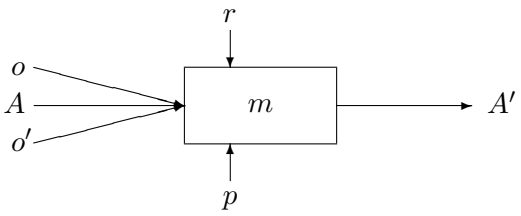

Figure 2.2: The matching process.

There have been many different matching methods proposed from various viewpoints (e.g., databases, information system, artificial intelligence). They take advantage of the various properties of ontologies (e.g., structures, data instances, semantics, or labels) and use techniques from different fields (e.g., statistics and data analysis, machine learning, automated reasoning, linguistics). These solutions share some techniques and tackle similar problems, but differ in the way they combine and exploit their results. Deliverable 2.2.3 [Euzenat *et al.*, 2004], provides an extensive

review of the state of the art of these methods.

2.2.2 Alignment structure and API

We have tried to provide a strict definition of the alignment structure so as to be able to use and reuse it in various situations. Given two ontologies O and O', alignments are made of a set of mappings between pairs of (simple or complex) entities $\langle e, e' \rangle$ belonging to O and O' respectively.

A correspondence (or mapping) is described as a quadruple:

$$\langle e, e', R, n \rangle$$

where:

- e and e' are the entities (e.g., formulas, terms, classes, individuals) between which a relation is asserted by the mapping;
- R is the relation, holding between e and e', asserted by the mapping. For instance, this relation can be a simple set-theoretic relation (applied to entities seen as sets or their interpretation seen as sets), a fuzzy relation, a probabilistic distribution over a complete set of relations, a similarity measure, etc.
- n is a degree of confidence in that mapping (notice, this degree does not refer to the relation R, it is rather a measure of the trust in the fact that the mapping is appropriate ("I trust 70% the fact that the mapping is correct/reliable/..."). The trust degree can be computed in many ways, including users' feedback or log analysis;

Having such a common format allows to compose results of different methods easily. For instance, it could help an agent in finding the relationships between its ontology and that of some other agent and use this to understand its messages.

In order to facilitate this we designed the Alignment API [Euzenat, 2004]. The API is a JAVA description of tools for accessing alignments in the format presented above. It is currently implemented on top of the OWL API and proposes the following services:

- Storing, finding, and sharing alignments;
- Piping alignments algorithms (improving an existing alignment);
- Manipulating (trimming and hardening);
- Generating processing output (transformations, axioms, rules in formats such as XSLT, SWRL, OWL, C-OWL);
- Comparing alignments.

Part of the interface of the API is presented in Table 3.1. The API can be used for producing transformations, rules or bridge axioms independently from the algorithm which produced the alignment. Several matching methods have been developed within this API.

2.2.3 Analysis

Ontology matching is a tool rather than a process that can help in ensuring interoperability.

It can be introduced within each agent in order to align the ontologies before interpreting and attempting to understand messages. However, this does not help with negotiating the correspondences between the terms known to an agent and this imposes heavy constraints on the agent design. Consequently, it is better to have some mechanism that does not have to embed alignment techniques. This kind of mechanism will be presented in the next section.

2.3 Ontology negotiation protocol

S. Bailin and W. Truszkowski (NASA / Knowledge Evolution, Inc.) proposed an Ontology Negotiation Protocol: ONP [Bailin and Truszkowski, 2002].

2.3.1 Approach

ONP deals with the very problem that we have to consider: allowing information agents to take advantage of distributed knowledge bases.

ONP is triggered when an agent B receives a message m from another agent A. B will invoke various mechanisms in order to understand the message and enrich his own ontology with acquired information. The protocol is based on four steps:

interpretation Consists for agent B of trying to interpret m. This involves checking for the definition of terms in its ontology. It can use external resources such as synonym dictionaries and check with A the corresponding terms. When the set of remaining unknown terms is larger than a particular threshold, B engages in a clarification dialogue with A.

clarification For each of the unresolved terms, A will propose some relationships (such as synonymy or subsumption) between this term and other terms in its ontology.

relevance evaluation Consists of evaluating the relevance of the obtained terms with regard to the current query answers. The agent will query its database and compare the similarity of documents to the query (in information retrieval terms, i.e., documents and queries are considered as a set of keywords). If the similarity is high, then the terms are considered as "aligned".

update Consists for each agent of updating its ontology in order to introduce the learnt knowledge.

This is a general description of the negotiation process which is managed by a finite state machine whose goal is to determine the actions of agents.

The process uses Wordnet as an external resource allowing to disambiguate unknown terms. Nowadays, synonym dictionaries are just one part of common matching algorithms.

2.3.2 Analysis

The objective of ONP is the understanding of messages. It negotiates the alignment term by term (instead of globally).

While the paper refers to ontologies, the terms are really taken as natural languages terms (through the attention made to synonyms for instance) and the messages look like natural language sentences. This process is relatively remote from semantic web technology in which each concept is identified by a term within some namespace (thus avoiding collision).

There are additional formal problems with this process: the negotiation is strongly connected to its context - one agent and one message (through relevance feedback). However, in the update phase, the result is included within the agents ontologies. The limit of the process is that each agent will have the same ontology made of some sort of union of all the terms and their relations.

Moreover, the relevance step does not seem cleanly defined since it will only consider the retrieved documents, not those which have not been retrieved.

This is to be contrasted with a world where agents keep their own ontologies, that they have been designed to reason with, while keeping track of the relations with other agents ontologies. In

addition, such a solution would allow one agent to take advantage of the alignments produced by other agents.

2.4 Mapping learning process

F. Wiesman, N. Roos et P. Vogt from MERIT/Infonomics have designed a mapping learning process (MLP) [Wiesman *et al.*, 2001].

2.4.1 Approach

The principle of the MLP is that of finding correspondences between instances of the concepts. The starting hypothesis of this approach is that it is not possible to find alignments by negotiation and that it is necessary to learn them by example.

This approach does not require to share concepts beforehand, but requires that instances (here instances are documents) are comparable. This is reasonable in information retrieval applications as considered in this paper. Ontologies are put in correspondence through the instances and the concepts they belong to. There is no need, in this case, to exchange more than concept identifiers. In particular, the concept hierarchy of the ontology of one agent is not known by the other agent.

The learning process is based on language games developed in [Steels, 1996] whose goal is to obtain some consensus communication framework starting from nothing. It consists of exchanging terms and evaluating them.

The process is as follows:

- an agent sends an instance of a concept to another agent,
- the receiver attempts to find the concept of which an instance has the maximum number of common words with the instance sent in the previous step,
- the process is reiterated

One of the agents can then establish correspondences between leaf concepts (or at least concepts which have proper instances). To that extent, it requires that the other agent sends the instances, along with the concepts they belong to. Over time the association between concepts can be reinforced (or weakened). The process stops when the learning agent considers that the correspondences are correct. This alignment is considered final. More than an alignment, it is usually a mapping from the ontology of the first agent to the ontology of the other one. The result is considered asymmetric.

Moreover, there are operators allowing to modify the instances so that one instance of one concept can be transformed in an instance of another concept in the other ontology.

2.4.2 Synthesis

This approach is a good general way of learning correspondences in ontologies. However, it suffers from several drawbacks:

- it requires the availability of some training set (which has often to be large);
- it takes a long dialogue time before learning the correspondences;
- there is no direct negotiation of the correspondences themselves.

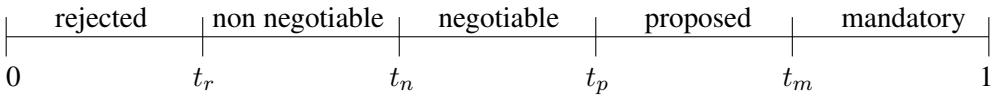

Figure 2.3: Characterisation of correspondence utility

	r	nn	n	p	m
r	R	R	R	R	X
nn	R	R	R	R	X
n	R	R	?	T	A
p	R	R	T	A	A
m	X	X	A	A	A

Table 2.3: Composition of MAFRA categories

2.5 MAFRA Ontology mapping negotiation

The MAFRA Ontology mapping negotiation [Silva *et al.*, 2005], aims at establishing the consensus between different agents which use the MAFRA alignment framework.

2.5.1 Approach

For accomodating agents with different utility for the correspondences, MAFRA defines a "utility" function (u) that depends on parameters representing the use context ($p_1, \ldots p_n$). The utility characterises a correspondence as a function of the confidence the system has in it and the parameters. This amounts to distributing the correspondences in five different classes depending on four thresholds, as shown in Figure 2.3.

The catagories are then used in a negotiation process which basically combines these categories according to Table 2.3. This classifies the correspondences into 5 categories:

A the correspondence is accepted;
T the correspondence is tentatively accepted (and will be decided in a further step);
? the decision is to be forwarded to the users.
R the correspondence is rejected;
X no consensus is reached: the negotiation failed;

The process of composing these threshold values is iterated after a meta-"utility" function is applied to the results. This meta-"utility" function has three purposes:

– decide that the effort devoted to resolution exceeds the expected benefit and to stop the negotiation;
– revise the thresholds of the "utility" function;
– decide globally on the tentatively accepted correspondences.

2.5.2 Analysis

The MAFRA Ontology mapping negotiation is the first attempt at solving the alignment negotiation problem with concrete measures that are independent from the external context (training set or common top-level ontology). It is highly dependent on the use of MAFRA, and the "utility" function is not precisely defined (as well as the meta-"utility" function). The jump from confidence to utility is at worst not very well argued, at best deeply tied to MAFRA.

2.6 Other related work

There are a number of other partially relevant works that do not solve exactly the same problem as the one considered here. In particular, in one of the most specific and relevant research events in the subject, MCN'2004 (Meaning Coordination and Negotiation Workshop at ISWC-2004, sponsored by Knowledge Web [Bouquet and Serafini, 2004]) many research papers on ontology coordination (mapping) where presented, none where presented about ontology negotiation.

2.6.1 Mutual online ontology alignment

[Wang and Gasser., 2002] propose something very similar to MLP but slightly more formalised. Their learning process is symmetric in that in step 2, the agent which receives the instance will reply with another instance of the same class: this will help agents converge towards some common understanding. Moreover, the agents do not learn correspondences between concepts, but instead modify their respective ontologies in order to fit those of the other agents. They use conceptual clustering in order to perform this step.

2.6.2 Optimal communication vocabularies

[van Diggelen *et al.*, 2004] tries to find the optimal communication vocabulary between two agents. This is in the (very easy) case in which each ontology is expressed in terms of some common grounding ontology. The paper defines the ideal communication ontology as:

- allowing the transfer of information without loss;
- minimises the size of messages;
- expressible in the common ontology;
- admitting no subset satisfying these conditions.

However, this approach assumes that agent ontologies are expressed using the same language. In a more recent work [van Diggelen *et al.*, 2005], the authors present a decentralised approach for agreeing on a grounding ontology, in a decentralised way. This approach however, has two main drawbacks, first of all it assumes that all agents will eventually comply to this grounding ontology, therefore all agents will eventually comply to a common conceptualisation of the domain. The second drawback concerns the process used for reaching consensus. This process is based on the assumption that one of the agents (the one that initiates the conversation) A_1 is capable of teaching a concept definition to the agent it is talking to A_2. That is, it is able to establish a mapping between the concept that is the object of a type of `inform` performative, and a concept belonging to the ontology known to A_2. The paper, however, fails to explain how this mapping is established.

2.6.3 DILIGENT methodology

DILIGENT [Tempich *et al.*, 2005] is a methodology for helping dealing with heterogeneous and evolving ontologies. It uses techniques similar to those presented here: agents provide arguments initially framed in the Rhetorical Structure Theory and Issue-bases information systems in order to track design choices in ontologies. However, the arguments are applied to ontology components rather than correspondences because the goal of the method is to find acceptable changes to those ontologies that have to be considered as consensual. Moreover, no evaluation framework (like in MAFRA or in the present report) is provided for evaluating arguments because this is the task of a committee. Finally, the arguments are formulated in order to support the process of consensus formation among ontology engineers, rather than to reach consensus on ontological definitions automatically.

2.6.4 Meaning negotiation for information retrieval

In the context of information retrieval, [Ermolayev *et al.*, 2005] proposed a strategy for meaning negotiation between a query evaluation engine and a mediator. The process consists of reducing some "semantic distance" between the two viewpoints represented by the ontologies (or rather the parts of their ontologies which are involved in answering the query). Negotiation proceeds by proposing correspondences which contribute to reducing this distance. It succeeds if the distance reaches a certain threshold, and fails when the threshold is not reached but agents have no further correspondence to propose. It is, however, not clear how an agent can evaluate the proposed correspondence and what is a good "semantic distance".

2.6.5 General drawbacks of the above approaches

There are common features of the above proposed approaches that are not very favourable to the openness and dynamics of the semantic web:

- all agents must have a common alignment mechanism;
- a training set is often necessary (realistic in information agents, not for web services);
- too much time is necessary for obtaining an alignment;
- common languages or ontologies are required.

We would like, in the following, to outline a solution to the meaning negotiation problem that overcomes these problems.

2.7 Negotiation approaches: bargaining and argumentation

In order for agents to fulfil their objectives, they need to interact. Different types of interaction mechanisms exist in the literature, such as coordination (in which agents arrange their individual activities in a coherent manner), collaboration (in which agents work together to achieve a common objective), and so on. One such interaction that is gaining increasing prominence in the agent community is Negotiation. Negotiation has been extensively studied by the agent community (see [Wooldridge, 2002] for more details), as a way to enable *cooperation* among agents. Loosely speaking, cooperation is the process by which computational agents choose to work together in order to achieve a common goal or to solve a common problem. Agents can be more or less

cooperative, and Franklin [Franklin and Graesser, 1997] provides a characterisation of multi-agent system that is based on the level of *cooperation* or *independence* exhibited by the agents. Figure illustrates this categorisation, where the classification is based on the extent of the intention to cooperate and on the presence of shared goals.

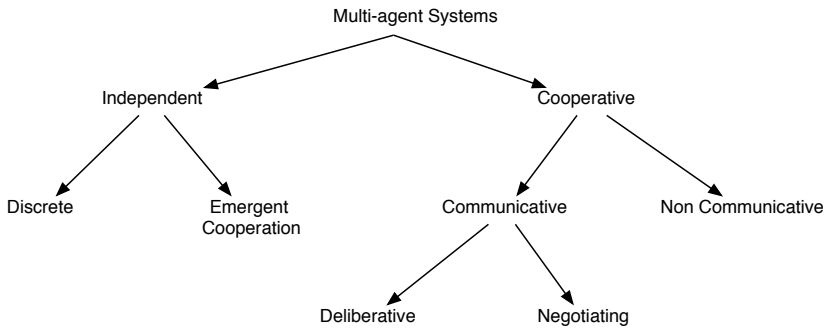

Figure 2.4: A classification of the types of agents

Franklin distinguishes between two types of cooperation: *deliberation* and *negotiation* depending on the intent involved in the communication. In more detail, in deliberative systems cooperation is achieved because agents jointly plan the actions to perform in order to achieve a common goal. Negotiating systems are systems where agents can compete with each other, to some extent. In other words, we can see negotiation as an *economic* approach, that is used in order to allocate some scarce resource, that is perceived to be valuable to one or more agents. In particular, we denote with *bargaining* a negotiation process that takes place between two agents only, that is the process by which two agents attempt to reach agreement on how to divide the benefits of their cooperation. It typically takes place in a series of negotiation steps, where each side puts forward a proposal. Much of the work on bargaining is based on game theory, and, although this approach has produced significant results, it assumes a number of limiting assumptions concerning the agents knowledge and utility functions, mainly related to how agents can influence one another in order to reach agreement, and how new issues can be introduced into an open negotiation.

Indeed, often agents have no authority over one another, and therefore the only possible way they have to influence the behaviour of their peers is to *persuade* them to act in a certain way. Persuading another agent usually consists of one agent convincing a second agent that the proposed course of action is consistent with the agent's goal, and that it might possibly be beneficial to the second agent. If the second agent disagrees it might be challenged to change its beliefs and preferences through some form of "justification" (*argument*) that might support or challenge a proposal. This is the object of argumentation-based negotiation[Sierra *et al.*, 1998].

In this deliverable we take the view that agents (be they software components, services, or human) need to reach an agreement on the terminology they need to use in order to communicate, and the process of forming consensus must be dynamic, and flexible. For this reason we choose to follow an argumentation-based approach for negotiating on meaning. Argumentation is inherently qualitative, and therefore better suited to reach agreements on the meaning of certain concepts in an ontology. Bargaining based approaches are more inherently more quantitative, as they are based on the evaluation of the utility for some agent of a certain proposal. The choice of an argumentation framework over a game theoretic one still allows us to model the notions of *gain* and *loss* that are usually expressed by an increase or decrease of the agent utility. Agents are self-interested and by

favouring a certain type of mapping over another an agent expresses its preferences on the possible course of actions, without having to disclose their utility functions.

In proposing an argumentation framework for meaning negotiation, we follow the structure of the proposal by Sierra and colleagues [Sierra *et al.*, 1998] and we outline the components of a formal model for the process of argumentation-based negotiation which can ultimately be used to negotiate in terms of possible alignments. We therefore concentrate on the *social aspects* of the negotiation process. We define a model that allows agents to make different types of arguments in support of their proposals, and we indicate how these arguments can be generated and interpreted by agents.

Chapter 3

Alignment negotiation protocol

From the example provided in the previous chapter, we identify the need for some service for providing ontology alignments that can be used by agents. In order to not commit to some agent architecture, these services will be considered as agents that communicate with other agents in the usual way.

In this chapter we present an alignment negotiation protocol (ANP) allowing agents to take advantage of web resources in order to find alignment between ontologies. The next chapter will consider the actual negotiation between agents.

This protocol does not take any decision on whether the alignments are stored for reuse or if they are computed on the fly. This allows to deal in the same way with alignments which have been designed statically by hand and alignments that are generated for the current case.

ANP is designed to be used whenever an agent needs it: when the agent receives a message to be understood he can start an alignment negotiation session and later resume the initial dialogue. The protocol uses the agent technology for exchanging messages (namely FIPA ACL). The protocol is described in some ad hoc manner that is easy to understand and can be directly transformed into some program ([Pecheur, 1997]).

ANP revolves around four entities that will be presented below:

Alignment services can be queried to match two ontologies or to translate a message according to an alignment; they are the main component described in Section 3.2;

Alignment libraries store alignments between ontologies; they will be considered here as special cases of alignment services;

Alignment protocol defines the way agents can communicate with alignment services; it is decribed in Section 3.4;

Directory of services and library enables the identification of alignment services; we will consider here that agents can use well-known facilities such as FIPA Directory Facilitator or UDDI [Clement *et al.*, 2005].

3.1 What's in an agent

The protocol imposes very few constraints on the agents. Indeed, one agent can receive a message that cannot be understood because it is expressed in an ontology O and the agent feels that it would be able to understand it in ontology O'. If the agent is able to find some alignment service S and to process the protocol, then it can have the message translated by the alignment service.

The additional constraint is that the agent is able to keep track of the dialogue corresponding to this protocol (this is very often omitted in multi-agent protocols, although very important if one needs to follow several dialogues at once). For that purpose, we use message identifiers.

Of course, any specially designed agent will be able to take more advantage of the protocol. This is the case if the agent can:

- record the surrogates for alignments that are returned by the alignment service;
- record the alignments themselves;
- record and process the programs used for translating messages or bridge axioms.

The agent can also produce by itself some alignment and ask the alignment service to store it.

3.2 Alignment API and alignment service

The alignment negotiation protocol is based on the existence of particular agents that are called *Alignment services*. These agents are able to perform a number of alignment tasks and offer them to the other agents. These tasks are summarised in Table 3.1.

Service	Syntax
Finding a similar ontology	$O' \Leftarrow Match(O, T)$
Align two ontologies	$A' \Leftarrow Align(O, O', A, P)$
Thresholding	$A' \Leftarrow Threshold(A, V)$
Generating code	$P \Leftarrow Render(A, language)$
Translating a message	$m' \Leftarrow Translate(m, A)$
Storing alignment	$n \Leftarrow Store(A, O, O')$
Suppressing alignment	$Delete(n)$
Finding (stored) alignments	$n \Leftarrow Find(O, O')$
Retrieving alignment:	$\langle O, O', A\rangle \Leftarrow Retrieve(n)$

Table 3.1: Servives provided by the alignment service

Most of these services correspond to what is provided by any implementation of the Alignment API [Euzenat, 2004]. In addition, the services must be able to test two predicates $reachable(O)$ and $conform(P)$ which respectively check if some ontology is reachable to the service and if a set of parameters is well-formed.

Alignment services must embed some structure that allows them to deliver adequately the offered services. These structures are presented in Table 3.2. They are the set of alignments stored in the service and indexed by the pair of ontologies (O, O'). Storing them has several purposes:

- being able to exchange only surrogates (identifiers) of the alignments instead of the whole alignments between agents;
- being able to reuse the alignment with several agents.

In fact the characteristics of each alignment (whether they are made by hand or by some particular algorithms for instance) are stored within the alignment. These alignments are indexed by ontology pairs and by surrogates allowing fast retrieving. To one surrogate corresponds only one alignment while for an ontology pair, there can be several such alignments.

Alignment services also know the set of languages in which they are able to deliver the alignments (these correspond to the renderers of the Alignment API).

Name	Content	Description	Status any agent	 alignment service
M	$\langle O, O', A \rangle$	Triples connecting two ontologies and their alignment	O	M
L	l	Set of supported export languages		M

Table 3.2: Data structures available in the alignment service (O=optional, M=mandatory)

Additional data structures can be embedded by the alignment service in case it wants to subcontract part of its work to other alignment servers. This is described in section 3.4.3.

3.3 Performatives

Alignment services and other agents will exchange messages in order to reach a satisfying alignment between two ontologies. For that purpose, they will exclusively use classical FIPA performatives as described in Table 3.3.

performative	How used in the protocol
request	Sender asks receiver to perform some action.
inform	Answer to some query.
query-if	Sender asks receiver to evaluate some query in some context.
failure	Sender notifies the failure of an action attempt.

Table 3.3: FIPA performatives used in the protocol.

Most of the time, these performatives will carry more precise messages expressed in what we call 'pseudo-performatives' (they are not performatives because they do not occur at the first level of a message but they are used to refine the meaning of a performative). Their meaning is defined in Table 3.4. They are expressed in this protocol language (ANP language).

3.4 Protocol definition

So far, we have described what structure and functions are available from alignment services and what messages they can send. The protocol specifies which message it is appropriate to send in response to some incoming message and in some situation.

The protocol is defined by a set of rules, as below:

(rulename)
$$\frac{a - m \rightarrow S}{\begin{array}{c} M \Leftarrow M \cup \{O\} \\ S - m' \rightarrow a \end{array}} \; O \notin M$$

Pseudo-performative	Meaning
$align(O, O', n, P)$	Sender asks to perform/return some ontology matching/alignment.
$render(n, l)$	Sender asks to return some translation program.
$translate(m, n)$	Senter asks to translate message m in function of the alignment identified by n.
$find(O, T)$	Sender asks to find an ontology (similar to another one).
$is - align(O, O')$	Sender asks if there exists an alignment between O and O'.
$unreachable(O)$	Ontology O is not reachable (network connectivity, access rights).
$unknown(n)$	No alignment is identified by n
$nonconform(P)$	The set of parameters (P) is non conform
$nomatch(O)$	No ontology matching ontology O has been found
$unsupported(l)$	Rendering in format (l) is not supported

Table 3.4: Extra pseudo-performatives embedded in the FIPA performatives (ANP language).

This rule stipulates that when receiving a message (m) an agent or service (S) can check some conditions ($O \notin M$) and, if they succeed, will perform the stipulated set of actions. These actions can be using the available operations (for services), including matching ontologies and storing the result ($M \Leftarrow M \cup \{O\}$), or sending messages ($S - m' \rightarrow a$).

All the rules respect the implicit contracts of answering messages with an :reply-with by the corresponding :in-reply-to and preserve the :protocol and and :conversation-id features. The content of all request and failure messages are written in the ANP :language depicted in Table 3.4, :protocol field is ANP and there are no :ontology nor :encoding.

Such rules can easily be translated in AUML but we find them easier to express this way.

3.4.1 Alignment services

The main function of the alignment service is to match two ontologies. It is invoked with the align request. The rule align-success defines what happens in such a case: the service checks that it can reach the ontologies, aligns them, stores the resulting alignment and sends back a surrogate for this alignment to the first agent.

(align-success)
$$\frac{a - request(align(O, O', n, P)) \rightarrow S}{\begin{array}{c}\langle O, O', A\rangle \Leftarrow Retrieve(n) \\ A' \Leftarrow Align(O, O', A, P) \\ n' \Leftarrow Store(O, O', A') \\ S - inform(n') \rightarrow a\end{array}} \begin{array}{c}\neg Find(O, O', n, P) \wedge \\ reachable(O) \wedge reachable(O') \wedge \\ conform(P) \wedge Retrieve(n) \neq \emptyset\end{array}$$

If some alignment is, however, stored in the service library, this service will directly get it and send it back to the agent:

(align-library)
$$\frac{a - request(align(O, O', n, P)) \rightarrow S}{\begin{array}{c}n' \Leftarrow Find(O, O', n, O) \\ S - inform(n') \rightarrow a\end{array}} Find(O, O', n, O) \neq \emptyset$$

Of course, based on some parameters, the alignment service can forward the task to some other alignment service (it can also be an alignment service broker that asks other services to align the ontologies and aggregates the results). This will be described in Section 3.4.3

The align-unreachable, align-unknown or align-nonconform rules define what happens when one of the ontologies is not reachable from the alignment service, when the initial alignment is unknown, or when the parameters required by the alignment method are not conform (for instance, if the agent asks to produce the alignment with some particular algorithm that is not available on the service).

(align-unreachable) $$\frac{a - request(align(O, O', n, P)) \rightarrow S}{S - failure(unreachable(O)) \rightarrow a} \neg reachable(O)$$

(align-unreachable) $$\frac{a - request(align(O, O', n, P)) \rightarrow S}{S - failure(unreachable(O')) \rightarrow a} \neg reachable(O')$$

(align-unknown) $$\frac{a - request(align(O, O', n, P)) \rightarrow S}{S - failure(unknown(n)) \rightarrow a} \neg Retrieve(n)$$

(align-nonconform) $$\frac{a - request(align(O, O', n, P)) \rightarrow S}{S - failure(nonconform(P)) \rightarrow a} \neg conform(P)$$

3.4.2 Search services

Alignment services can also be used for finding an ontology similar to another one. As a matter of fact, computing alignments often amounts to computing a distance between the elements of an ontology. This service can be invoked through the find performative. It can be quite useful when some ontology is not reachable from some alignment service and another service can provide a closer ontology. When invoked in this way the alignment service will return an ontology close to the initial one.

(search-success) $$\frac{\begin{array}{c} a - request(find(O, T)) \rightarrow S \\ O' \Leftarrow Match(O, T) \end{array}}{S - inform(O') \rightarrow a} reachable(O) \wedge Match(O, T) \neq \emptyset$$

If the service does not find some reasonably similar ontology, it will report this failure:

(search-void) $$\frac{a - request(find(O, T)) \rightarrow S}{S - failure(nomatch) \rightarrow a} reachable(O) \wedge Match(O, T) = \emptyset$$

Of course, this invocation can fail if the ontology is not reachable from the service.

(search-unreachable) $$\frac{a - request(find(O,T)) \rightarrow S}{S - failure(unreachable(O)) \rightarrow a} \neg reachable(O)$$

It is possible to query the service for the availability of some alignment on two ontologies. The service will answer with the alignment surrogate if available (rule query-align-success) or with an error message (rule query-align-empty).

(query-align-success)
$$\frac{a - query - if(is - align(O,O')) \rightarrow S}{\begin{array}{c} n \Leftarrow Find(O,O',\_,\_) \\ S - inform(n) \rightarrow a \end{array}} \begin{array}{c} Find(O,O',\_,\_) \neq \emptyset \wedge \\ reachable(O) \wedge reachable(O') \end{array}$$

(query-align-empty)
$$\frac{a - query - if(is - align(O,O')) \rightarrow S}{S - failure(nomatch) \rightarrow a} \begin{array}{c} Find(O,O',\_,\_) = \emptyset \wedge \\ reachable(O) \wedge reachable(O') \end{array}$$

As previously, the service will report a failure when the ontologies are out of its reach:

(query-align-unreachable) $$\frac{a - query - if(is - align(O,O')) \rightarrow S}{S - failure(unreachable(O)) \rightarrow a} \neg reachable(O)$$

(query-align-unreachable) $$\frac{a - query - if(is - align(O,O')) \rightarrow S}{S - failure(unreachable(O')) \rightarrow a} \neg reachable(O')$$

Some agent – or another alignment service – can also ask the alignment service to store an alignment of its own (or communicated by some other agent):

(store-alignment) $$\frac{a - request(store(O,O',A)) \rightarrow S}{\begin{array}{c} n \Leftarrow Store(O,O',A) \\ S - inform(n) \rightarrow a \end{array}}$$

3.4.3 Subcontracting

Alignment services communicating with each other need to be able to locate other services (this can be achieved through some data structure or better through some directory service like UDDI) and to be able to record the queries they receive because message answering will become asynchronous.

Name	Content	Description
Q	$\langle s, i, i', q, S' \rangle$	$s - q(i) \rightarrow S$ has been forwarded to S' with message surrogate i'
R	$\langle s, n, n', S' \rangle$	ontology surrogate n for agent s corresponds to surrogate s' for S'
C	$querytype \rightarrow \{S'\}$	services S' have the capability to answer given query types

Table 3.5: Additional data structures available for subcontracting queries (all are optional for normal operation)

This mandates to store the queries and their origins as well as surrogate rewriting.

In order to deal with subcontracting of queries, the services find (in Q) another service supposed to handle the query and forward the query to this service with a fresh message identifier. Agents only use redirection when they cannot answer the query themselves. This is controlled by setting $C(q)$ to the empty set for the queries that must not be forwarded. Moreover, in order to render the result more deterministic, each failure sending rule must ensure that redirection is not possible by adding a $C(q) = \emptyset$ clause.

For surrogate rewriting, the function $R(\cdot)$ and $R^{-1}(\cdot)$ use the structure R to replace the surrogates in query arguments. The ! operator generates a new surrorgate and the $surr(\cdot)$ function indicates if some content is a surrogate.

The first rule redirects some request that the service cannot handle to another service that can.

$$\text{(redirect)} \qquad \frac{a - request(q(x)\ reply - with:\ i) \rightarrow S}{\begin{array}{c} Q \Leftarrow Q \cup \{\langle a, i, !i', q(x), S' \rangle\} \\ S - request(q(R(x))\ reply - with:\ i') \rightarrow S' \end{array}}\ S' \in C(q)$$

Another such rule deals with the query-if queries of the protocol.

The return of these queries is simply handled by forwarding the result to the initial issuer of the query.

$$\text{(handle-return)} \qquad \frac{S' - inform(y\ reply - to:\ i') \rightarrow S}{\begin{array}{c} Q \Leftarrow Q - \{\langle a, i, i',, S' \rangle\} \\ S - inform(R^{-1}(y)\ reply - to:\ i) \rightarrow a \end{array}}\ \langle a, i, i',, S' \rangle \in Q, \neg surr(y)$$

But in case the answer is a new surrogate identifying some alignments, this surrogate is stored in R and a new local surrogate is generated.

$$\text{(handle-return)} \qquad \frac{S' - inform(y\ reply - to:\ i') \rightarrow S}{\begin{array}{c} Q \Leftarrow Q - \{\langle a, i, i',, S' \rangle\} \\ R \Leftarrow R \cup \{\langle a, !y', y, S' \rangle\} \\ S - inform(R^{-1}(y)\ reply - to:\ i) \rightarrow a \end{array}}\ \langle a, i, i',, S' \rangle \in Q, surr(y)$$

This applies to the forwarding of the failures as well.

(failure-return)
$$\frac{S' - failure(y\ reply - to:\ i') \rightarrow S}{\begin{array}{c} Q \Leftarrow Q - \{\langle a, i, i',, S'\rangle\} \\ S - failure(R^{-1}(y)\ reply - to:\ i) \rightarrow a \end{array}}\ \langle a, i, i',, S'\rangle \in Q$$

We assume that the services obey this protocol and thus do not generate $inform$ and $failure$ messages that do not correspond to some previous query in Q.

3.4.4 Translation services

Alignments can be directly used from the service instead of being used within the agent. The agent can ask for the service to translate some message from one ontology to the other and get the translated message back (rule translate-success).

(translate-success)
$$\frac{a - request(translate(M, n)) \rightarrow S}{\begin{array}{c} \langle O, O', A\rangle \Leftarrow Retrieve(n) \\ m' \Leftarrow Translate(m, A) \\ S - inform(m') \rightarrow a \end{array}}\ Retrieve(n) \neq \emptyset$$

This can fail if the servive does not know the required alignment:

(translate-unknown)
$$\frac{a - request(translate(M, n)) \rightarrow S}{S - failure(unknown(n)) \rightarrow a}\ Retrieve(n) = \emptyset$$

3.4.5 Export alignment

Instead of directly translating a message, it is possible to retrieve the alignment itself in its external format (RDF) or in some directly usable format. Such a format can be a translation programme (in XSLT for instance), a set of logic formulas (axioms asserting the correspondences in the alignments), or some set of rules that can be used for importing the content of a message (in SWRL, C-OWL or other formalisms). The agent will use the render performative with, as arguments, the surrogate of some alignment and the language in which the agent wants the alignment. The get-processor-success rule takes care of this.

(get-processor-success)
$$\frac{a - request(render(n, l)) \rightarrow S}{\begin{array}{c} \langle O, O', A\rangle \Leftarrow Retrieve(n) \\ P \Leftarrow Render(A, l) \\ S - inform(P\ language:\ l) \rightarrow a \end{array}}\ l \in L \wedge Retrieve(n) \neq \emptyset$$

Of course this action can fail if the alignment is unknown (rule get-processor-unknown) or if the language is not suppported (rule get-processor-failure).

(get-processor-unknown) $$\frac{a - request(render(n, l)) \rightarrow S}{S - failure(unknown(n)) \rightarrow a}\ Retrieve(n) = \emptyset$$

(get-processor-failure) $$\frac{a - request(render(n, l)) \rightarrow S}{S - failure(unsupported(l)) \rightarrow a}\ l \notin L$$

3.4.6 Initialising dialogue

Here are all the messages that can initiate the dialogue within the protocol (i.e., which do not need a previous message):

$$\begin{aligned} a - request(align(O, O', n, P)) &\rightarrow S \\ a - request(find(O, T)) &\rightarrow S \\ a - query - if(is - align(O, O')) &\rightarrow S \\ a - request(store(O, O', A)) &\rightarrow S \\ a - request(translate(M, n)) &\rightarrow S \\ a - request(render(n, l)) &\rightarrow S \end{aligned}$$

3.5 Example

We provide below a complete example of the use of this protocol.

3.5.1 Agents

The scenario presented here involves four agents: two agents a and b are communicating but agent a uses ontology O while agent b uses O'. b will call two alignment services c and d, with c being a powerful aligner with a restricted access to environment of b (it cannot access O') and d having a broader access.

3.5.2 Dialogue

// Agent a is looking for a book and asks agent b
$a-$query-ref(:ontology O
:language RDQL
:content "SELECT x WHERE x O:autobiography http://www.bertrandrussell.com"
:reply-with 1)$\rightarrow$ b
// Agent b does not understand ontology O and asks service c to align it with O'
$b-$request(:content align(O,O',$\emptyset$,$\emptyset$) :reply-with 1)$\rightarrow$ c
// Service c cannot reach ontology O'
$b \leftarrow$failure(:in-reply-to 1 :content unreachable(O'))$-c$ (align-unreachable)

// Agent b asks d to find a similar ontology
b−request(:content find(O',m) :reply-with 2)→ d
O'' ⇐Match(O',T)
// Service d found O''
b ←inform(:in-reply-to 2, :content O'')−d (search-success)
// Agent b asks service d to align O' with O''
b−request(:content is-align(O',O'') :reply-with 3)→ d
s ⇐ Find(O',O'',∅,∅)
// Service d had already stored such an alignment and returns it
b ←inform(:in-reply-to 3, :content s')−d (align-library)
// Agent b asks service c to align O with O''
b−request(:content align(O,O'',∅,∅) :reply-with 4)→ c
A ⇐ Align(O,O'',∅,∅)
s ⇐ Store(O,O'', A)
// Service c computes the alignment
b ←inform(:in-reply-to 4, :content s)−c (align-success)
// Agent b asks service c to translate the message with the found alignment
b−request(:content translate(m, s) :reply-with 5)→ c
$\langle O, O'', A\rangle$ ⇐ Retrieve(s)
m ⇐ Translate(m, A)
b ←inform(:in-reply-to 5 (translate-success)
:content "SELECT x
WHERE x O':biography http://www.bertrandrussell.com.
x O':author http://www.bertrandrussell.com.")−c
// Agent b asks service d to translate the result with the O' to O'' alignment
b−request(:content translate(m', s') :reply-with 6)→ d
$\langle O', O'', A'\rangle$ ⇐ Retrieve(s')
m'' ⇐ Translate(m', A')
b ←inform(:in-reply-to 6 (translate-success)
:content "SELECT x
WHERE x rdf:type O':biografia.
x dc:subject http://www.bertrandrussell.com.
x dc:creator http://www.bertrandrussell.com.")−d
// The returned query is evaluated by agent b
QueryResult(m'') ⇒ x=http://isbn.org/2-436-4428-1
// which returns the answer to agent a
a ←reply-ref(:content "x=http://isbn.org/2-436-4428-1" :in-reply-to 1)−b
// a is satisfied and wants to know the publisher of the book
a−request-ref(:content "http://isbn.org/2-436-4428-1 O:publisher x" :reply-with 2)→ b
// b had not recorded the alignment surrogate and asks it to c
b−request(:content align(O,O",∅,∅) :reply-with 7)→ c
// which only has to retrieve it from its store
s ⇐ Find(O,O",∅,∅)
b ←inform(:content s :in-reply-to 7)−c (query-align-success)
// b asks c for a mapping program in order to translate the messages by itself
b−request(:content render(s, C-OWL) :reply-with 8)→ c

// but c cannot deliver this format
$b \leftarrow$failure(:content unsupported(C-OWL) :in-reply-to 8)$-c$ (get-processor-failure)
// so b asks for another one
$b-$request(:content render(s, XSLT) :reply-with 9)$\rightarrow c$
$\langle O, O'', A\rangle \Leftarrow$ Retrieve(s)
$P \Leftarrow$ Render(A, XSLT)
$b \leftarrow$inform(:content P :language XSLT :in-reply-to 9)$-c$ (get-processor-success)
// which is delivered and used by b to translate the message
$m' \Leftarrow P(m)$
// The translation goes once again through d
$b-$request(:content translate(m', s') :reply-with 10)$\rightarrow d$
$\langle O', O'', A'\rangle \Leftarrow$ Retrieve(s')
$m'' \Leftarrow$ Translate(m', A')
$b \leftarrow$inform(:in-reply-to 10 (translate-success)
:content "SELECT x
WHERE http://isbn.org/2-436-4428-1 dc:publisher x")$-d$
// and the query is processed by b
QueryResult(m'') $\Rightarrow$ "x=http://www.example.com/#Routledge"
// which returns the result
$a \leftarrow$reply-ref(:content "x=http://www.example.com/#Routledge" :in-reply-to 2)$-b$

3.6 Protocol properties

The protocol is very simple and all its power comes from the alignment API and associated algorithms. There are however a few properties that can be exhibited.

Proposition 1 (Exhaustivity)**.** *The protocol is exhaustive: for each received message by an agent, there is at least one rule to trigger.*

Proof. By enumeration of the conditions of each rules. □

However, the dual property of exclusivity (that there is only one rule for each case that triggers) is not true: the protocol is non deterministic because if there are many failures, it can trigger any of the failure rules. It can be made deterministic however, by strictly ordering the activation of these rules.

Another interesting property is that of unicity of the result. This is not guaranteed due to the alignment algorithms which have no way to choose between two equally good alignments.

Chapter 4

Argumentation framework

So far, we have only provided the opportunity for agents to obtain an alignment, and to use this alignment in order to understand messages. We can as well assume that agents are able to generate such an alignment using an independently defined decision-making process. There is a salient difference between the work on alignment, that deals with how to find an alignment, and the work done here in multi-agent systems, dealing with how to share such an alignment. What is considered in this context is not the internal beliefs or decision making process, but the external commitments of an agent towards some correspondences.

However, such alignments may not be satisfactory for both parties. So, in order to communicate, agents can enter a negotiation and try to find a common alignment. In order to carry out this negotiation, agents can use argumentation: agents argue on the selection or rejection of a particular correspondence in the framework of some alignment [Rahwan *et al.*, 2004; Reed and Norman, 2004]. Argumentation about correspondences and properly negotiating them requires that agents are able to exchange arguments in favour or against particular correspondences when challenged. Here the goal of the process is obviously not that agents adopt a common ontology, but that they commit to a particular set of correspondences between their heterogeneous ontologies.

Accomplishing this task requires:

- arguments and, in particular, arguments relevant to alignments (§4.1),
- argumentation performatives enabling the expression of the will to challenge and justify arguments, and a protocol for arguing (§4.2) - which states not only how to propose arguments and what is an appropriate reply to some utterance, but also what to do when some agreement is reached,
- and finally a logic, or any other means, for evaluating arguments and selecting the whole alignment on the basis of this evaluation (§4.3).

Fortunately, for each of these requirements, there are results that can be adapted to the negotiation of alignments.

It is noteworthy, that [Reed and Norman, 2004] mentions common ontology as a prerequisite for argumentation. Since we use argumentation for finding this common ontology, it is clear that this is not a prerequisite (as long as the argumentation does not apply to the content).

4.1 Arguments

A priori, any logical formula can be an argument for or against another one (see §4.3). However, in the specific application of negotiating ontology alignments, we can have a more specific argument language than just resorting to any logic language. The general arguments for accepting or rejecting some correspondences are well known since they are the basis of matching algorithms. In this section, we consider these arguments and their relations with proposed correspondences.

4.1.1 Types of arguments

There can be many arguments for accepting or rejecting some correspondence. A rough classification is the following:

semantic (M): the sets of models of some expressions do or do not compare;
internal structural (I): the two entities share more or less internal structure (e.g., the value range or cardinality of their attributes);
external structural (S): the set of relations of two entities with other entities do or do not compare;
terminological (T)): two names of entities share more or less lexical features;
extensional (E): the sets of instances of two entities do or do not compare.

Let's take an example: We start with two ontologies O and O', here in description logics but this is easily expressed in OWL:

$$O = \{Micro - company = Company \sqcap \leq_5 employee\}$$
$$O' = \{SME = Firm \sqcap \leq_{10} associate\}$$

in addition to which the following initial alignment is given:

$$A = \{\langle Company, Firm, =, .89\rangle, \langle associate, employee, \sqsubseteq, 1.0\rangle\}$$

The system has to judge the following statement:

$$\gamma = \langle micro - company, SME, \sqsubseteq, .97\rangle$$

One agent can argue in favour of γ:

a_1 that the alignment plus the two ontologies entail the correspondence (semantic);
a_2 that all the known *micro-companies* on one side are *SME* on the other side (and not vice versa); (extensional);

or it can argue against it:

a_3 that the two names *micro-companies* and *SME* are not alike by any string distance, and they are not synonyms in WordNet (terminological);
a_4 that the only features they share are *associate* and *employee* and they have different domains and cardinalities (structural).

These are basically the same kind of arguments that are used for justifying alignment algorithms. In fact, this is relatively natural. They are easier to be used as arguments than for alignments because one has just to find the argument for one pair of entities while alignments are obtained globally.

4.1.2 Arguments in some OWL correspondences

As can be seen from the previous example, the type of argument depends on the correspondence and especially on the type of entities (properties, concepts, etc.) and the relation considered by the correspondence. We will provide below a list of arguments for or against correspondences between OWL entities.

For example, given a correspondence described as $\langle e, e', \equiv, n\rangle$, where e and e' are classes, the list of arguments in favour may be:

- their labels are similar **(terminology)**
- their identifiers are similar **(terminology)**
- their instances are similar **(extensional)**
- a low/high fraction of their instances are similar **(extensional)**
- their properties are similar **(internal structural)**
- their sub-concepts are similar **(external structural)**
- their super-concepts are similar **(external structural)**
- their siblings (i.e. children of parents) are similar **(external structural)**
- the alignment plus the two ontologies entails the correspondence **(semantic)**

Given a correspondence described as $\langle e, e', \equiv, n\rangle$, where e and e' are properties, the list of arguments in favour may be:

- their labels are similar **(terminology)**
- their identifiers are similar **(terminology)**
- the instances connected by two properties are the same **(extensional)**
- their sub-properties are similar **(external structural)**
- their super-properties are similar **(external structural)**
- their domain and range are the same **(internal structural)**

Given a correspondence described as $\langle e, e', \equiv, n\rangle$, where e and e' are individuals, the list of arguments in favour may be:

- their labels are similar **(terminology)**
- their identifiers are similar **(terminology)**
- their parent concepts are similar **(structural)**
- the properties that link the two individuals are the same **(structural)**

Table 4.1 summarizes a number of arguments and the correspondences in OWL they can support or attack. These can be used in causal arguments. However, other arguments may be used in particular situations. These can be general arguments like 'I prefer to have this correspondence' (an authority argument) which can be used without specificity with regard to the alignments. On the contrary, in a particular context, an agent may want to use a particular general formula for justifying a particular correspondence. This is however application dependent and cannot be investigated generally here.

4.1.3 Preference arguments

Preference arguments are often useful, particularly when one wants to argue that one correspondence is better than another one. It should be based on the simple comparison of homogeneous criteria.

For instance, it can be argued, on an extensional basis that one class is better than another for matching a particular third class because it shares more instances with this last one:

$$prefer(E(c) \cap E(c') \subseteq E(c) \cap E(c''), \langle c, c', \equiv, 1.0\rangle, \langle c, c'', \equiv, 1.0\rangle)$$

In fact, any of the arguments above can be turned into a preferential argument which compares the similarity of the considered feature.

4.1.4 Global arguments

There are arguments that do not apply to one correspondence but to an alignment as a whole. These arguments occur, for instance when one has some (similarity) measure between the entities to align and an optimization criterion applying to the alignment as a whole. This is the case of global similariy

For instance, imagine that the similarity matrix between the elements of one ontology $O = \{c_1, c_2\}$ and another one $O' = \{c'_1, c'_2\}$ is :

$$\begin{pmatrix} 9 & 7 \\ 6 & 1 \end{pmatrix}$$

based on this, the agent could argue in favour of $\langle c_1, c'_1, =, 1.\rangle$ because it has the highest similarity. However, a counter argument is that in case of a 1:1 alignment, this choice does not yield an optimal global similarity because it will constrain the choice of $\langle c_2, c'_2, =, 1.\rangle$ yielding a global similariy of 10, while the other choice $\{\langle c_1, c'_2, =, 1.\rangle, \langle c_2, c'_1, =, 1.\rangle\}$ provides a similarity of 13.

4.2 Alignment argumentation protocol

Arguments are used for negotiating the correspondences that must be part of the alignments. In the typical agent way, arguments are exchanged within a well defined protocol. The protocol specifies how negotiation should take place, including the flow of messages to be used. This includes the interaction protocol as well as other rules of the dialogue.

In this section, we further define the protocol as an instance of the framework of [McBurney and Parsons, 2004]. We first introduce the various performatives that have been defined for exchanging arguments (§4.2.1) then we define the principles along which a dialogue should unfold (§4.2.2) before providing the precise specification of the protocol rules (§4.2.3).

4.2.1 Meaning argumentation performatives

Agents need to be able to express that they argue against or in favour of some statement. This can be understood independently from the statement and the argument and is thus a good candidate speech act.

Just as action negotiation has been built into the FIPA ACL, there are good reasons for introducing argumentation performatives. [McBurney and Parsons, 2004] recently introduced such an extension that will be presented here. It is based on a propositional content language K from which the set A of arguments is taken. An argument a can support some assertion k (noted $a \vdash^+ k$) or attack it ($a \vdash^- k$). The new performatives are the following:

$assert(k)$ the sender asserts a statement $k \in K$. This means that the sender is committed to argue in favour of it, providing a justification for k if required subsequently by another participant.

$retract(k)$ the sender who had uttered $assert(k)$ or $justify(a, k)$ can withdraw this statement with the performative $retract(k)$ or $retract(a \vdash^{+} k)$, respectively. This removes the earlier obligation on sender to justify k or a if questioned or challenged.

$question(k)$ the sender asks the receiver to justify k (which the receiver is supposed to be committed to). The sender is not obliged to justify the question utterance.

$challenge(k)$ the sender asks the receiver to justify k. In contrast to a question, with this locution, the sender is obligated to provide a justification for not asserting k, for example an argument against k, if questioned or challenged.

$justify(a, k)$ the sender justifies k by argument $a \in A$ (i.e., $a \vdash^{+} k$ or $a \vdash^{-} k$). The performative $justify(a \vdash^{+} k, k)$ and $justify(a \vdash^{-} k, k)$ is similarly defined.

The performative $assert(.)$ forms the basis of argument. The set of performatives is relatively limited even with regard to the set of objects and relations that can be defined in the simple IBIS system. But it will be sufficient for our purposes.

It is noteworthy that with the arguments being part of the content language, it is not necessary to have specific performatives for justifications supporting or attacking other justifications. For our purpose, we distinguish between supporting arguments and attacking arguments, introducing two shorthands:

$$support(a, k) \equiv justify(a \vdash^{+} k, k)$$

and

$$contest(a, k) \equiv justify(a \vdash^{-} k, k)$$

In the context of alignment negotiation, we will distinguish between the language of arguments which will be defined in the next section and the propositional language which is initially restricted to correspondences as they have been defined in Section 2.2. However, in order to counter arguments, it is necessary to add all of the argument language A to the propositional language (so that we finally have $A \subseteq K$).

Moreover, in order to introduce preferences between correspondences and argue for or against them, we add some specific performatives that could be rewritten in a stronger propositional language as follows:

$$prefer(a, k, k') \equiv justify(a \vdash^{+} k < k', k < k')$$

In summary, negotiation is achieved through the exchange of a particular proposition using a shared communication language like FIPA ACL extended with the proposed performatives (summarized in Table 4.2).

4.2.2 Outline of argument dialogues

We assume that the dialogues take place between two agents, and that agents interact using the argumentation performatives. A dialogue is initiated by an agent who proposes an alignment and each correspondence it contains. So agent a will start by asserting some correspondence:

$$a - assert(k) \rightarrow b$$

or some alignment:

$$a - assert(A) \rightarrow b$$

meaning that agent a is committed to support alignment A (and each correspondence it contains). against any attack which might came from the other agent. Of course, this agent may have retracted correspondences that it does not support from some alignment. When making use of the alignment, the agent is committed to use only those correspondences that it has asserted and not retracted.

If the other party does not agree with some correspondence k, the assert can be challenged ($challenge(k)$) or questioned ($question(k)$). A challenge or a question can be replied to with an argument a ($support(a, k)$). In the challenge case, support must be replied with an argument for not asserting k ($contest(a, k)$). Such counter arguments can be responded to as any other argument. Contests can be responded to by retracting the challenged proposition ($retract(k)$).

Therefore, a dialogue resembles a game, in which the players successively make *moves* according to the protocol, by introducing arguments.

An utterance of some performative by one agent may cause an update of its commitment store. The commitment store of an agent is the set of formulas the agent is committed to endorse according to its utterances, during the dialogue. For instance, the fact that agent a has expressed $assert(A)$, resulted in storing all the correspondences of A in its commitment store and all other agents knowing that a is bound to follow these correspondences. Since agents have to exchange two kinds of information – knowledge and preferences – the commitment stores will have two parts: $CS.Kb$, will contain knowledge and $CS.Pref$, will contain preference ($CS = CS.Kb \cup CS.Pref$).

Once performed, these dialogue moves are added via commitment stores to the agents background knowledge, that is,we assume that all these moves are perfectly perceived by all the agents.

We specify all the dialogue moves that may be legal in some circumstances, namely the possible legal follow-ups after a dialogue move $a - p(S) \rightarrow b$, where p is one of the performatives presented in 4.2.1, a and b are, respectively, the sender and the receiver of the performative, and S is the content of the performative. a makes the argument we are interested in and its defenders and b makes the counter-arguments or defeaters. A dialogue can be viewed as a sequence of speech-acts made by agents:

Definition 1. *An argument dialogue between two agents a and b is a nonempty sequence of dialogue moves $p_1, \ldots, p_n$ such that:*

- *$\forall i \geq 0$ if p_i is uttered by agent a (resp. b), then p_{i+1} (if any) is uttered by agent b (resp. a);*
- *$CS_0(a) = \emptyset$ and $CS_0(b) = \emptyset$. The commitment stores CS are empty at step 0;*
- *For any p_i, p_j, if $i \neq j$ then $S_i \neq S_j$, where S_i and S_j are, respectively, the subjects of the moves p_i and p_j.*

By the first condition, agents utter alternately in a dialogue. The second condition says that the commitment stores are empty at the beginning of the dialogue. The last condition prevents, for example, that an agent repeatedly asks the same question. The set of dialogue moves (or performatives) and the protocol together will define the set of legal moves available to an agent at any time. With this background, we can present the set of dialogue moves that we will use.

This is the classical definition of a dialogue. Paolo Bouquet remarked the second condition was desirable in practice if agents have commitments from other dialogue or other sources. This is a reasonable assumption that we can endorse (one of the reasons for having the commitment store empty is to be able, for instance, to retract only commitments from the current dialogue).

4.2.3 Argument dialogue moves

For each move, we give what we call rationality rules and update rules. These are based on the rules suggested by [Maudet and Evrard, 1998]. In particular, we will describe for each move how the move updates the commitment stores (the update rules), the legal next steps for the other agent (the dialogue rules) and the rationality rules as the precondition for a move.

We suppose that agent a addresses a move to agent b. The dialogue moves are the following:

assert(k) where $k \in K$. Initially it is restricted to an alignment and their correspondences.

rationality: Check if there is an acceptable argument for the statement k.
dialogue: the other agent can respond with:
1. $question(k)$
2. $challenge(k)$

update: $CS.Kb_i(a) = CS.Kb(a)_{i-1} \cup \{k\}$ and $CS.Kb_i(b) = CS.Kb(b)_{i-1}$

question(k) where $k \in K$

rationality: There is no rationality condition.
dialogue: the other agent can only respond with $support(x, k)$ and $contest(x, k)$ where x is the argument supporting or attacking k.
update: $CS.Kb_i(a) = CS.Kb_{i-1}(a)$ and $CS.Kb_i(b) = CS.Kb_{i-1}(b)$

challenge(k) where $k \in K$

rationality: Check if there is an acceptable argument for the statement $\neg k$.
dialogue: the other agent can only respond with $support(x, k)$ and $contest(x, k)$ where x is the argument supporting or attacking k.
update: $CS.Kb_i(a) = CS.Kb_{i-1}(a)$ and $CS.Kb_i(b) = CS.Kb_{i-1}(b)$

support(a, k) where $k \in K$ and x is the argument supporting k

rationality: Check if the related argument is acceptable.
dialogue: the other player can make any move
update: $CS.Kb_i(a) = CS.Kb_{i-1}(a)$ and $CS.Kb_i(b) = CS.Kb_{i-1}(b)$

contest(x, k) where $k \in K$ and x is the argument supporting k

rationality: Check if the related argument is acceptable.
dialogue: the other player can make any move
update: $CS.Kb_i(a) = CS.Kb_{i-1}(a)$ and $CS.Kb_i(b) = CS.Kb_{i-1}(b)$

retract(k) where $k \in K$

rationality In response to an assertion, check if there is no acceptable argument for k.
dialogue The other player can make any move except retract.
update $CS.Kb_i(a) = CS.Kb_{i-1}(a) \setminus \{k\}$ and $CS.Kb_i(b) = CS.Kb_{i-1}(b)$

prefer(x, k, k') where $k, k' \in K$ and x is the argument supporting the preference of k over k'

rationality: the agent checks if the related argument is acceptable.
dialogue the other agent can make any move.
update $CS.Pref_i(a) = CS.Pref_{i-1}(a) \cup \{x\}$ and $CS.Pref_i(b) = CS.Pref_{i-1}(b)$

These dialogue moves define precisely what an agent can do with regard to argumentation. However, most of these moves require to evaluate the arguments before making the move. This is achieved with the help of an argumentation system as described below.

4.3 Argumentative logic for evaluating arguments

So far, the framework has only presented the syntactic moves (i.e., the definition of a well-formed dialogue). In order to behave during the negotiation, agents must be able to evaluate arguments. We propose below to take advantage of the work on argumentative logic in order to determine what can be an acceptable argument and when an argument should be accepted.

The logic of argumentation will tell how to choose between the various correspondences in function of the provided arguments [Amgoud *et al.*, 2000]. In the context of agents, it is very often built into the protocol itself, like in the persuasion machine [Reed and Norman, 2004]), which will have an important impact on the way agents use the protocol. However, having it outside the protocol allows to be able to change one theory for another.

4.3.1 Logic of argumentation

In this section, we recall the basis of the theory of argumentation based on contextual preference of [Amgoud and Parsons, 2001], which extends the work of Dung in dealing with preferences between arguments. We assume that the knowledge base is equipped with a (partial or total) preordering representing the preferences of the agent.

Let L be a propositional language, we denote with $\Sigma \subseteq L$ the knowledge base of an agent. An agent is able to consider arguments from different viewpoints called contexts. Let C be the set of these contexts. Formulas in Σ may be ordered by a preference relation $\ll_c$ depending on the considered context $c \in C$ ($x \ll_c y$ meaning that y is prefered to x in c). We can define the argumentation framework as:

Definition 2 (Argumentation system)**.** *An argumentation system (AF) is a quadruple* $\langle \Sigma, C, \sqsupset, \{\ll_i\}_{i \in C} \rangle$*, where:*

- Σ *is the knowledge base;*
- C *is a set of contexts;*
- $\sqsupset$ *is a preference relation on* $C \times C$ ($c \sqsubset c'$ *meaning that* c' *is preferred over* c)*, and*
- $\{\ll_i\}_{i \in C}$ *is a set of order relations depending on the contexts in* C*.*

Each agent is equipped with its own argumentation system that it uses in order to evaluate the arguments that are submitted to it.

We can define the notion of an argument.

Definition 3 (Argument)**.** *An* argument Arg *is a pair* $Arg = \langle H, h \rangle$ *where* h *is a formula of* L *and* H *is a subset of* Σ *such that:*

1. H *is consistent;*
2. h *is a logical consequence of* H*;*
3. H *is a minimal subset of* Σ*, from which* h *can be inferred.*

H and h are called, respectively, the *support* and the *conclusion* of Arg. The set of arguments over a set of formulas Σ is denoted by $Arg(\Sigma)$.

An argument is attacked if and only if there exists an argument for the negation of an element of its support. $Arg(\Sigma)$ will contain arguments which *attack* each other:

Definition 4 (Attack)**.** *Let* $Arg_1 = (H_1, h_1)$ *and* $Arg_2 = (H_2, h_2)$ *be two arguments of* $Arg(\Sigma)$*,* Arg_1 attacks Arg_2 *iff* $\exists h \in H_2$ *such that* $h \equiv \neg h_1$

The *defense* relation is defined as a dual relation of *attack*. Therefore, argumentation is based on the construction of arguments and counter-arguments, the comparison of these various arguments and finally the selection of the arguments that are considered to be acceptable.

So far, there are some differences between this logical framework and the current state of alignment negotiation:

- we did not consider supports as sets so far;
- the support for arguments as given before is not strictly based on logical consequence;
- the conclusions of arguments are not logical formulas but correspondences.

However, this can be overcome in the following way:

- support can straightforwardly extended to sets;
- one can use a argumented consequence relation ($\Vdash$) and Table 4.1 provides exactly such a relation;
- the notion of an attack can be redefined with the notion of contradictory correspondences ($\perp$).

Therefore, we replace the notion of $attack$ with the following definition:

Definition 5 (Correspondence attack)**.** *Let $Arg_1 = (H_1, h_1)$ one argument of $Arg(\Sigma)$ and $m = (e, e', R, \_)$ be a correspondence, Arg_1* attacks *m iff $h_2 \perp R(e, e')$.*

One difference from the traditional attack relation is that this one is symmetric. However, by using this definition of attack, it is possible to take advantage of the usual formulation.

Definition 6 (Preferred argument)**.** *Let $Arg_1 = (H_1, h_1)$ and $Arg_2 = (H_2, h_2)$ be two arguments of $Arg(\Sigma)$, Arg_1 is preferred to Arg_2 iff $\exists y \in H_2$ such that $\forall x \in H_1, y \ll x$*

An argument defends itself if it is preferred to those arguments which seek to attack it, and a set of arguments can defend a lone argument by attacking all those arguments which the lone argument cannot defend itself against:

Definition 7 (Self-defense)**.** *Let Arg_1 and Arg_2 be two arguments of $Arg(\Sigma)$, such that Arg_2 attacks Arg_1 then Arg_1 defends itself against Arg_2 iff:*

1. *$\exists c \in C$ such that $Arg_1 \gg_c Arg_2$, and*
2. *$\forall c' \in C$ such that $Arg_1 \ll_{c'} Arg_2$ then $c' \sqsupset c$*

Otherwise Arg_1 does not defend itself against Arg_2.

Definition 8 (Group defense)**.** *A set of arguments S defends Arg_1 iff $\forall Arg_2 \in Arg(\Sigma)$ such that Arg_2 attacks Arg_1 and Arg_1 does not defend itself against Arg_2 then $\exists Arg_3 \in S$ such that Arg_3 attacks Arg_2 and Arg_2 does not defend itself against Arg_3.*

The grounded semantic of AF is defined as the least fixpoint of a function $F : Arg(\Sigma) \longrightarrow Arg(\Sigma)$ such that:

$$F(S) = \{A \in Arg(\Sigma) | A \text{ is defended by } S\}$$

Now, we can define the set of acceptable arguments for an argumentation system AF:

Definition 9 (Acceptable arguments)**.** *The set of acceptable arguments for an argumentation system* $\langle \Sigma, C, \sqsupset, \{\ll_i\}_{i \in C} \rangle$ *is:*

$$\bigcup_{i=0}^{+\infty} F^i(\emptyset)$$

such that :

$$F(S) = \{A \in Arg(\Sigma) | A \textit{ is defended by } S\}$$

An argument is *acceptable* if it is a member of the acceptable set (which contains all the non attacked and self defending arguments). If the argument (H, h) is *acceptable*, we talk of there being an acceptable argument for h, and we say that the proposition h is acceptable to an agent that has an acceptable argument for it. An acceptable argument is one which is, in some sense, proven since all the arguments which might undermine it are themselves undermined.

[Simon Parsons and Wooldridge, 2004] shows that all these elements can be used to construct argumentation dialogues.

4.3.2 Strategy of a dialogue

Now that we have defined the protocol as the set of legal moves available to an agent, the next issue is: when there is a choice, what should an agent say? In fact, an agent can choose a move from the set of legal moves, and the choice of a particular move is the result of his strategy. A strategy in an argumentation dialogue specifies what utterances to make in order to bring about some desired outcome (e.g. to persuade the counterpart to perform a particular action). In order to define a strategy, we need to take into account several factors, such as the nature of the previous move, the protocol, the goal of the agents, etc. Indeed, the agent must select the kind of act but also the content to expose, and this depends on the belief of the agent and on current state of the dialogue, among other factors. In consequence, a strategy can be captured in different levels. In particular, [Amgoud and Maudet, 2002] defines three levels:

- definition of same agent profile;
- choosing to build or destroy a strategy;
- choosing some appropriate argumentative content.

The first level is the agent profile, which needs to be completed with some other level. In general, a classification of a number of strategies which reflect the agent profile, is the following:

- Agreeable: *accept* whenever possible
- Disagreeable: only *accept* when no reason not to
- Open-minded: only *challenge* when necessary
- Argumentative: *challenge* whenever possible
- Elephant's child: *question* whenever possible

At the next level, we will consider that the agent can adopt either a build or a destroy strategy, where a strategy is selected towards any statement from the participants' commitment stores.

- **b-strategy** consists of defending some facts in its commitment store.
- **d-strategy** consists of attacking some commitments in the opponent' commitment store.

The last level concerns the problem of the (argumentative) content of the act. In order to define this level, we define a level of acceptable arguments.

Definition 10 (Levelled set of acceptable arguments)**.** *The set of acceptable arguments at level l for an argumentation framework AF is* $\underline{S}^l = \bigcup_{i=0,\cdots,l} F_{i\geq 0}(\emptyset)$

Definition 11 (Level of acceptable arguments)**.** *The level l at which the argument is acceptable in an argumentation framework AF is given by the lacc function defined as:*
$lacc\colon A(\Sigma) \rightarrow N$ *such that:*

1. *if* $A \in C_{attack}$ *then* $lacc(A) = 0$
2. *if* $A \in C_{attack,\sqsubset}$ *then* $lacc(A) = 1$
3. *if* $A \in S^l$ *and* $A \notin S^{l-1}$ *then* $lacc(A) = l + 1$

Definition 12 (Strengths of an argument)**.** *Let A and B be two arguments of* $\underline{S}$*. A is considered stronger than B iff* $lacc(A) < lacc(B)$.

The stronger an argument is, the more grounded is the conclusion supported by this argument. We define now the intuition behind the decision of whether to choose a **b-strategy** or **d-strategy** when the agent has the initiative: it is rational to select a **b-strategy** if there is some sufficiently acceptable argument available to support some fact in its commitment store. The sufficiently acceptable level is defined at level 1 by the agent profile (depending on its attitude) and the level of acceptability of the facts is computed at level 3.

In this section, we propose an example of a strategy process, involving the three levels. Consider an argumentation dialogue between an agent P, with a level of prudence p_a, and an agent C. We denote the commitment stores for the agents, respectively, CS_P and CS_C. We suppose that the agent P has adopted a current strategy **c-strategy** $\in \{$**b-strategy**, **d-strategy**$\}$. When he has the initiative, the agent can trigger the following algorithm:

1. if **c-strategy**=**b-strategy** then consider 2 and then 3, else consider 3 and then 2.
2. if the agent has some acceptable argument A such that:
   - $A = (H, p) \in \underline{S}$ and $p \in CS_P$
   - $lacc(A) = l'$ and $l' < p_a$

   Then the agent adopts a **b-strategy** and utters *assert(p)*
3. if the agent has some acceptable argument A such that
   - $A = (H, p) \in \underline{S}$ and $\neg p \in CS_C$
   - $lacc(A) = l'$ and $l' < p_a$

   then the agent adopts a **d-strategy** and utters *challenge(p)*
4. if the agent has no argument for some fact p in the opponent's commitment store, he adopts a **d-strategy** and utters *question(p)*
5. else the agent selects p in the opponent's commitment store for which he has some aceptable argument A such that:
   - $A = (H, p) \in \underline{S}$ and $p \in CS_C$
   - $\nexists p'\colon p' \in CS_C \wedge A' = (H', p') \in \underline{S}$ and $lacc(A) < lacc(A')$

   then the agent adopts a **d-strategy**

The algorithm captures the intuition that an agent will explore the position of the other party, in order to find a weak point in its line of argumentation.

4.4 Example

We give a example of a dialogue conducted under the protocol between two agents, labeled respectively with C and P, using the example in 4.1.1. The agent C starts the dialogue asserting a alignment A between two ontologies O and O' (the agent C is committed to support the alignment A and each correspondence it contains). The two ontologies O and O', expressed in description logic, are:

$$O = \{Micro - company = Company \sqcap \leq_5 employee\}$$
$$O' = \{SME = Firm \sqcap \leq_{10} associate\}$$

The alignment A is:

$$A = \{\langle Company, Firm, =, .89\rangle, \langle employee, associate, \sqsubseteq, 1.0\rangle, \quad \langle micro - company, SME, \sqsubseteq, .97\rangle\}$$

The three correspondences are denoted, respectively, with γ_1, γ_2 and γ_3.
The set of of arguments in favour of γ_1 are:

a_1 all the known *Company* on one side are *Firm* on the other side and vice versa (extensional); $E(Company) \subseteq E(Firm), E(Firm) \subseteq E(Company)$

a_2 the two names *Company* and *Firm* are synonyms in WordNet (terminological); $label(Company) \approx_d label(Firm)$

The set of of arguments in favour of γ_3 are:

a_3 the alignment plus the two ontologies entail the correspondence (semantic); $M(micro - company) \models M(SME)$

a_4 all the known *micro-companies* on one side are *SME* on the other side (and not vice versa) (extentional); $E(micro - company) \subseteq E(SME)$

and the counter-arguments are:

a_5 the two names *micro-companies* and *SME* are not alike by any string distance, and they are not synonyms in WordNet (terminological); $label(micro - companies) \not\approx_d label(SME)$

a_6 that the only features they share are *associate* and *employee* and they have different domains and cardinalities (structural). $S(micro - companies) \cap S(SME) \neq \emptyset$

A possible dialogue between C and P is the following:

//The agent C is committed to support the alignment
$C-$assert(:content A :reply-with 1)$\rightarrow P$
//The agent P asks to justify the correspondence γ_1 (P does not have counter-argument)
$C \leftarrow$ question(:content γ_1 :reply-with 2) - P
// The agent C justifies the correspondence γ_1 with the arguments a_1 and a_2
$C-$support(:content $a_1, a_2 \vdash^+ \gamma_1$:in-reply-to 2)$\rightarrow P$
//The agent P asks to justify the correspondence γ_3 (P is ready to justify the opposite)
$C \leftarrow$ challenge(:content γ_3 :reply-with 3) - P
// The agent C justifies the correspondence γ_3 with the arguments a_3 and a_4

$C-$support(:content $a_3, a_4 \vdash^+ \gamma_3$:in-reply-to 3)$\rightarrow P$
// The agent P contests the correspondence γ_3 with the counter-arguments a_5 and a_6
$C-$contest(:content $a_5, a_6 \vdash^- \gamma_3$:in-reply-to 3)$\rightarrow P$
// The agent C retracts the correspondence γ_3
$C-$retract(:content γ_3 :in-reply-to 3)$\rightarrow P$

4.5 Properties

There are some properties that need be shown and checked. A desirable theoretical property of the protocol is that it is *nonconcurrent*, namely that at most one dialogue move is generated at any time. This can be proved by construction. Another property is *exhaustivity*, namely if the protocol guarantees that at least one such admissible argument exists. A property, taken from [Amgoud and Parsons, 2001] is the following:

Proposition 2. *If two agents P and C, equipped with a argumentation framework AF_P and AF_C, start a argument dialogue using the set of perfomatives P_i in which P moves first, and S is the possible set of arguments generated, then:*

- *$\forall x \in S$ in the set of acceptable arguments of either AF_P or AF_C;*
- *If $x \in S$ is the set of acceptable arguments of AF_P, it is in the set of acceptable arguments of AF_C.*

These results show the *soundness* of the dialogues and the *termination*. Indeed, if the dialogue ends with an argument being acceptable to P, then it is also acceptable to C (P *persuades* C).

Id	Corresp.	$\vdash^?$	Argument	Comment
S1	$\langle e, e', \equiv, \_\rangle$	+	$S(e) \cap S(e') \neq \emptyset$	Entities have common neighbours (e.g., super-entities, sibling-entities, etc.)
S2	$\langle e, e', \equiv, \_\rangle$	-	$S(e) \cap S(e') = \emptyset$	Entities do not have common neighbours
S3	$\langle e, e', \sqsubseteq, \_\rangle$	+	$S(e) \subseteq S(e')$	e neighbours are included in those of e'
S4	$\langle e, e', \sqsubseteq, \_\rangle$	-	$S(e') \subseteq S(e)$	e' neighbours are included in those of e
S5	$\langle e, e', \perp, \_\rangle$	+	$S(e) \cap S(e') = \emptyset$	Entities do not have common neighbours
S6	$\langle e, e', \perp, \_\rangle$	-	$S(e) \cap S(e') \neq \emptyset$	Entities have common neighbours
I1	$\langle c, c', \equiv, \_\rangle$	+	$properties(c) \cap properties(c') \neq \emptyset$	Classes have common properties
I2	$\langle c, c', \equiv, \_\rangle$	-	$properties(c) \cap properties(c') = \emptyset$	Classes do not have common properties
I3	$\langle c, c', \sqsubseteq, \_\rangle$	+	$properties(c) \subseteq properties(c')$	c properties are included in those of c'
I4	$\langle c, c', \sqsubseteq, \_\rangle$	-	$properties(c') \subseteq properties(c)$	c' properties are included in those of c
I5	$\langle c, c', \perp, \_\rangle$	+	$properties(c) \cap properties(c') = \emptyset$	Classes do not have common properties
I6	$\langle c, c', \perp, \_\rangle$	-	$properties(c) \cap properties(c') \neq \emptyset$	Classes have common properties
I7	$\langle p, p', \equiv, \_\rangle$ $\langle p, p', \sqsubseteq, \_\rangle$	+	$range(p) \approx range(p')$	Properties have similar range
I8	$\langle p, p', \equiv, \_\rangle$ $\langle p, p', \sqsubseteq, \_\rangle$	-	$range(p) \not\approx range(p')$	Properties do not have similar range
I9	$\langle p, p', \perp, \_\rangle$	+	$range(p) \not\approx range(p')$	Properties have dissimilar range
I10	$\langle p, p', \perp, \_\rangle$	-	$range(p) \not\approx range(p')$	Properties have similar range
E1	$\langle e, e', \sqsubseteq, \_\rangle$	+	$E(e) \subseteq E(e')$	e instances are included in those of e'
E2	$\langle e, e', \sqsubseteq, \_\rangle$	-	$E(e') \subseteq E(e')$	c' instances are included in those of c
E3	$\langle e, e', \equiv, \_\rangle$	+	$E(e) \subseteq E(e'), E(e') \subseteq E(e)$	e instances are included in those of e' and vic.
E4	$\langle e, e', \equiv, \_\rangle$	-	$E(e) \cap E(e') = \emptyset$	Entities do not have common instances
E5	$\langle e, e', \perp, \_\rangle$	+	$E(e) \cap E(c') = \emptyset'$	Entities do not have common instances
E6	$\langle e, e', \perp, \_\rangle$	-	$E(e) \cap E(c') \neq \emptyset'$	Entities have a common instance
T1	$\langle e, e', \equiv, \_\rangle$ $\langle e, e', \sqsubseteq, \_\rangle$	+	$label(e) \approx_d label(e')$	Entities' labels are similar
T2	$\langle e, e', \equiv, \_\rangle$ $\langle e, e', \sqsubseteq, \_\rangle$	-	$label(e) \not\approx_d label(e')$	Entities' labels are dissimilar
T3	$\langle e, e', \sqsubseteq, \_\rangle$ $\langle e, e', \sqsubseteq, \_\rangle$	+	$URI(e) \approx_d URI(e')$	Entities' URIs are similar
T4	$\langle e, e', \equiv, \_\rangle$ $\langle e, e', \sqsubseteq, \_\rangle$	-	$URI(e) \not\approx_d URI(e')$	Entities' URIs are dissimilar
T5	$\langle e, e', \perp, \_\rangle$	+	$label(e) \not\approx_d label(e')$	Entities' labels are dissimilar
T6	$\langle e, e', \perp, \_\rangle$	-	$label(e) \approx_d label(e')$	Entities' labels are similar
T7	$\langle e, e', \perp, \_\rangle$	+	$URI(e) \not\approx_d URI(e')$	Entities' URIs are dissimilar
T8	$\langle e, e', \perp, \_\rangle$	-	$URI(e) \approx_d URI(e')$	Entities' URIs are similar

Table 4.1: Correspondences and arguments in OWL.

Performative	**How used in the protocol**
$assert(k)$	the sender asserts a statement $k \in K$
$retract(k)$	the sender does not support k anymore
$question(k)$	the sender asks the receiver to justify k
$challenge(k)$	the sender asks the receiver to justify k (he is obligated to justify $\neg k$)
$support(a, k)$	the sender supports k by argument a
$contest(a, k)$	the sender contests k by argument a
$prefer(a, k, k')$	the sender prefers k' than k by argument a

Table 4.2: Argumentation performatives

Conclusion

This report has investigated in depth the support for agents using different ontologies to interoperate.

First, we have presented an alignment negotiation protocol (ANP) that allows agents to take advantage of web resources in order to find alignments between ontologies. ANP, which uses the FIPA ACL technology for exchanging messages, is designed to be used whenever an agent needs it: when the agent receives a message to be understood he can start an alignment negotiation session and later resume the initial dialogue. This protocol enables any agent to obtain ontology alignments from specialised services taking advantage of the work carried out in workpackage 2.2.

Then, we have explored the problem of negotiating changes when alignments do not satisfy both parties. For that purpose we defined an argumentation-based framework which allows agents to argue about the correspondences and to evaluate the arguments in an adequate argumentative logic. The arguments provided by agents are specifically adapted to arguing about correspondences. We gave a relatively large library of such arguments (it can be extended though). The possible argumentation steps (or moves) are defined through the performatives introduced in [McBurney and Parsons, 2004]. Finally, the argumentative logic is based on the work by [Amgoud and Parsons, 2001], it defines which arguments are acceptable for an agent (so that it could concede or not the proposals of other agents).

This work has been presented in the context of agents with relatively strong reasoning capabilities, however nothing prevents most of it from being transposed to negotiation between semantic web services. It should also be possible to use it in order to help human agents to negotiate the alignments they need.

The first perspective of this work is the implementation of these features in an operational application. Some of these implementations are underway. This will certainly lead to updating the available library of arguments. Another useful direction should be the design of a global criterion for evaluating the proposed argumentation protocol (in particular, is it achieving an optimal compromise with regard to this criterion). This would certainly require a refinement of the preference relations.

Index

Index of rules

Bibliography

[Amgoud and Maudet, 2002] Leïla Amgoud and Nicolas Maudet. Strategical considerations for argumentative agents (preliminary report). In *Proc. 9th InternationalWorkshop on Non-Monotonic reasoning (NMR-2002). Special session on Argument, dialogue, decision*, pages 399–407, 2002.

[Amgoud and Parsons, 2001] Leïla Amgoud and Simon Parsons. Agent dialogues with conflicting preference. In *International Workshop on Agent Theories, Architectures and Languages (ATAL-2001)*, pages 190–205, 2001.

[Amgoud *et al.*, 2000] Leïla Amgoud, Simon Parsons, and Nicolas Maudet. Arguments, dialogue and negotiation. In *Proc. 14th European Conference on Artificial Intelligence, Berlin (DE)*, pages 338–342, 2000.

[Bailin and Truszkowski, 2002] Sidney Bailin and Walt Truszkowski. Ontology negotiation: How agents can really get to know each other, 2002.

[Bouquet and Serafini, 2004] Paolo Bouquet and Luciano Serafini, editors. *ISWC 2004 workshop on Meaning Coordination and Negotiation (MCN-04)*, 2004.

[Bouquet *et al.*, 2004] Paolo Bouquet, Jérôme Euzenat, Enrico Franconi, Luciano Serafini, Giorgos Stamou, and Sergio Tessaris. Specification of a common framework for characterizing alignment. deliverable D2.2.1, Knowledge web NoE, 2004.

[Clement *et al.*, 2005] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. Universal description, discovery and integration version 3.0.2. Specification, OASIS, 2005.

[Ermolayev *et al.*, 2005] Vadim Ermolayev, Natalya Keberle, Wolf-Ekkehard Matzke, and Vladimir Vladimirov. A strategy for automated meaning negotiation in distributed information retrieval. In *Proc. 4th ISWC, Galway (IE)*, pages 201–215, 2005.

[Euzenat *et al.*, 2004] Jerome Euzenat, Thanh Le Bach, Jesús Barrasa, Paolo Bouquet, Jan De Bo, Rose Dieng-Kuntz, Marc Ehrig, Manfred Hauswirth, Mustafa Jarrar, Rubén Lara, Diana Maynard, Amedeo Napoli, Giorgos Stamou, Heiner Stuckenschmidt, Pavel Shvaiko, Sergio Tessaris, Sven Van Acker, and Ilya Zaihrayeu. State of the art on ontology alignment. deliverable D2.2.3, Knowledge web NoE, 2004.

[Euzenat, 2004] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd international semantic web conference, Hiroshima (JP)*, pages 698–712, 2004.

[Finin *et al.*, 1993] Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald MacKay, James MacGuire, Richard Pelavin, Stuart Shapiro, and Chris Beck. Draft specification of the KQML agent communication language. Technical report, DARPA Knowledge Sharing Effort, 1993. ftp://ftp.cs.umbc.edu/pub/ARPA/kqml/papers/kqml.ps.

[FIPA0037, 2002] FIPA0037. FIPA ACL communicative act library specification. Technical report, FIPA, 2002. http://www.fipa.org/specs/fipa00037.

[FIPA0061, 2002] FIPA0061. FIPA ACL message structure specification, 2002. http://www.fipa.org/specs/fipa00061.

[Franklin and Graesser, 1997] S. Franklin and A. Graesser. Is it an agent, or just a program? In *Proceedings of ATAL96*, pages 21–36. Springer, 1997.

[Huget *et al.*, 2003] Marc-Philippe Huget, Bernhard Bauer, Jim Odell, Renato Levy, Paola Turci, Radovan Cervenka, and Hong Zhu. FIPA modeling: interaction diagrams. Technical report, Foundation for Intelligent Physical Agents, 2003. http://www.auml.org/auml/documents/ID-03-07-02.pdf.

[Maudet and Evrard, 1998] N. Maudet and F. Evrard. A generic framework for dialogue game implementation. In *Proc. 2nd Workshop on Formal Semantics and Pragmatics of Dialogue, University of Twente, The Netherlands*, 1998.

[McBurney and Parsons, 2004] Peter McBurney and Simon Parsons. Locutions for argumentation in agent interaction protocols. In Rogier van Eijk, Marc-Philippe Huget, and Frank Dignum, editors, *International Workshop on Agent Communication, New-York (NY US)*, volume 3396, pages 209–225, 2004.

[Pecheur, 1997] Charles Pecheur. Specification and validation of the co4 distributed knowledge system using lotos. In Iyad Rahwan, Pavlos Moraitis, and Chris Reed, editors, *Proc. 12th IEEE International Conference on Automated Software Engineering, Incline Village (NE US)*, 1997.

[Rahwan *et al.*, 2004] Iyad Rahwan, Sarvapali Ramchurn, Nicholas Jennings, Peter McBurney, Simon Parsons, and Liz Sonenberg. Argumentation-based negociation. *The knowledge engineering review*, 18(4):343–375, 2004.

[Reed and Norman, 2004] Chris Reed and Timothy Norman, editors. *Argumentation Machines: New Frontiers in Argument and Computation*. Kluwer , Dordrecht (NL), 2004.

[Sierra *et al.*, 1998] C. Sierra, N. R. Jennings, P. Noriega, and S. Parsons. A framework for argumentation-based negotiation. In *Proceedings of ATAL97*, pages 177–192. Springer-Verlaag, 1998.

[Silva *et al.*, 2005] Nuno Silva, Paulo Maio, and João Rocha. An approach to ontology mapping negotiation. In *Proc. K-Cap workshop on Integrating Ontologies, Banff (CA)*, pages 54–60, 2005.

[Simon Parsons and Wooldridge, 2004] Peter McBurney Simon Parsons and Mike Wooldridge. The mechanics of some formal inter-agent dialogues. In *Advances in Agent Communication*, pages 329–348, 2004.

[Smith, 1980] Reid Smith. The contract net protocol: high level communication and control in a distributed problem solver. *IEEE transactions on computer*, 29(12):1104–1113, 1980.

[Steels, 1996] Luc Steels. Emergent adaptive lexicons. In *From Animals to Animats 4: Proc. 4th International Conference On Simulating Behavior, Cambridge (MA US)*, 1996.

[Tempich *et al.*, 2005] Christoph Tempich, Helena Sofia Pinto, York Sure, and Steffen Staab. An argumentation ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT). In *Proc. 2nd ESWC, Crete (GR)*, pages 241–256, 2005.

[van Diggelen *et al.*, 2004] Jurriaan van Diggelen, Robbert Beun, Frank Dignum, Rogier van Eijk, and John-Jules Meyer. Optimal communication vocabularies and heterogeneous ontologies. In Rogier van Eijk, Marc-Philippe Huget, and Frank Dignum, editors, *International Workshop on Agent Communication, New-York (NY US)*, volume 3396, pages 76–90, 2004.

[van Diggelen *et al.*, 2005] Jurriaan van Diggelen, Robbert Beun, Frank Dignum, Rogier van Eijk, and John-Jules Meyer. A decentralized approach for establishing a shared communication vocabulary. In Jurriaan van Diggelen, Virginia Dignum, Ludger van Elst, and Andreas Abecker, editors, *AAMAS 05 workshop on Agent Mediated Knowledge Management)*, pages 1–13, 2005.

[Wang and Gasser., 2002] Jun Wang and Les Gasser. Mutual online ontology alignment. In *AAMAS OAS workshop*, 2002.

[Wiesman *et al.*, 2001] F. Wiesman, N. Roos, and P. Vogt. Automatic ontology mapping for agent communication. Research memorandum, MERIT-Infonomics, Maastricht (NL), 2001.

[Wooldridge, 2002] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons, 2002.

Related deliverables

A number of Knowledge Web deliverables are clearly related to this one:

Project	Number	**Title** and relationship
KW	D2.2.1	**Specification of a common framework for characterizing alignment** provides the framework which defines the notion of alignment.
KW	D2.2.3	**State of the art on ontology alignment** provides a panorama of many of the techniques that can be used both for computing alignments and evaluating arguments.
KW	D2.2.6	**Specification of delivery alignment format** provides a first description of the alignment format that is returned by the protocol and its implementation.
KW	D2.2.7	**Analysis of knowledge transformation and merging techniques** describes the rendering actions that are used by the alignment agent for providing programs using alignments.
KW	D2.4.3	**State of the art on agent-based services** provides an overview in the areas of reaching agreements, communication, and collaboration in multi-agent systems.